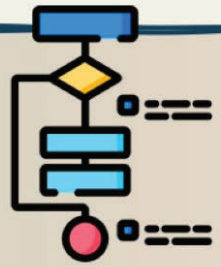
recap

Taylor and Maclaurin Series

True vs. approximation errors

- True percent relative error, $\varepsilon_t = \frac{\textit{true value} - \textit{approximation}}{\textit{true value}} \times 100\%$
- In most applications, the true value is not known. Instead, approximate errors
- $\varepsilon_a = \frac{\textit{current approximation} - \textit{previous approximation}}{\textit{current approximation}} \times 100\%$



- Iterative approach, iterate until a convergence
- Repeated until stopping criterion is satisfied: $|\varepsilon_a| < \varepsilon_s$
- Setting ε_s, for results to be correct to at least n sig figs: $\varepsilon_s = (0.5 \times 10^{2-n})\%$

The Taylor series

Provides a means to predict a function value at one point in terms of the function value and its derivative at another point

The n^{th} order Taylor approximation of the function f at x, centered at $x = a$, is

$$f(x) = \sum_{n=0}^{\infty} f^{(n)}(a)\frac{(x-a)^n}{n!} = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

- **Know ...**
- how to **construct** a Taylor polynomial, given a function f or its derivative
- how to **evaluate** the n^{th} term or the n^{th} order approximation at x
- determine the **coefficient** or the **derivative** of a specific term

Commonly use Maclaurin series Taylor centered at x=a=0

$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$	$-\infty < x < \infty$
$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots$	$-\infty < x < \infty$
$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \ldots$	$-\infty < x < \infty$
$\tan^{-1} x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} \ldots$	$-1 < x < 1$
$\ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$	$-1 < x < 1$
$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + x^4 + \cdots$	$-1 < x < 1$
$\frac{1}{1+x} = \sum_{n=0}^{\infty} (-1)^n x^n = 1 - x + x^2 - x^3 + x^4 + \cdots$	$-1 < x < 1$
$(1+x)^k = \sum_{n=0}^{\infty} \binom{k}{n} x^n = 1 + kx + \frac{k(k-1)}{2!}x^2 + \frac{k(k-1)(k-2)}{3!}x^3 + \cdots$	$-1 < x < 1$

- **Know ...**
  - how to read and evaluate these series
  - find the value of x to apply the formula, i.e. x has to satisfy the interval of convergence
    - ex. calculate $\ln 25$, or the square root of 5.2

try IT!

on Finex

We will upload the following to Finex

- ❑ PowerPoint slides (.pdf)
- ❑ Python files (.ipynb)
- ❑ These are the same files as those provided to you in My Courses

Office Hours

Friday 1:30 – 3:30pm, Lab203
(overlapped with your English classes)

- Catch up and do not give up
- Use the time in class to do homework (hand calculations and programming)
- Try practice problems in the lectures

ROOTS: BRACKETING METHODS

Textbook (Python) Part II Chapter 5

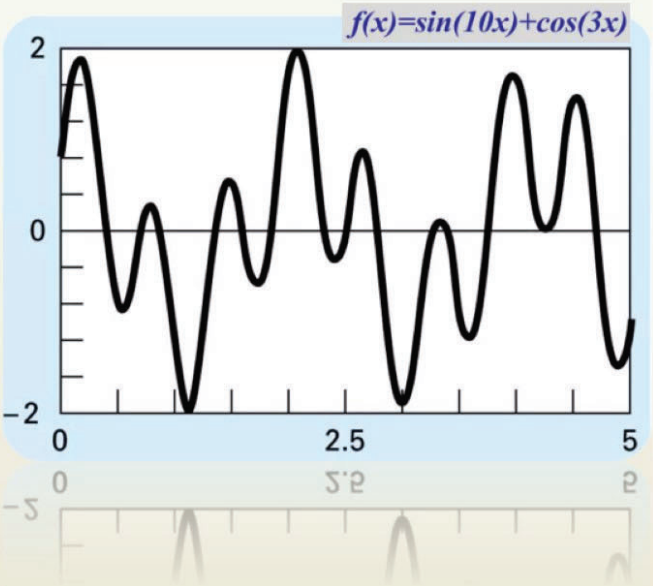
Why numerical methods? Why an approximation?

- **Want** to solve for the roots (find x)
  But it is difficult or impractical to find
  So, we mathematically approximate it

- **Easy:** $ax^2 + bx + c = 0$
  Solve analytically: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

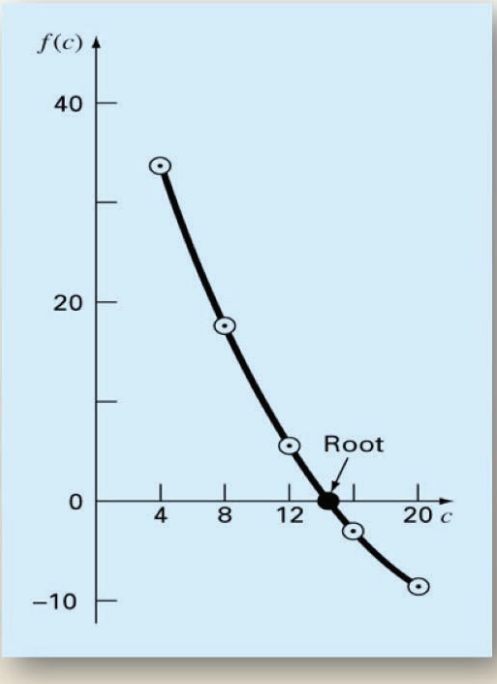
- **Not so easy:** $x = ?$
  $ax^5 + ax^4 + cx^3 + dx^2 + ex + f = 0$

- **How about this:** $\sin 10x + \cos 3x = 0$
  Instead, how about we **approximate** it?



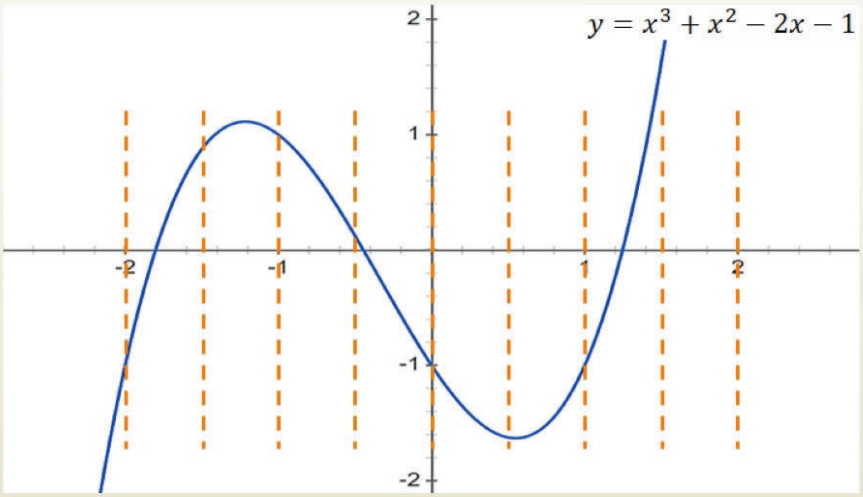
Bracketing Methods

- Two initial guesses for the root are required. These guesses must "bracket" the root, that is, they must be on either side of the root.
- E.g. two guesses can be $x_l = 12$ and $x_u = 16$



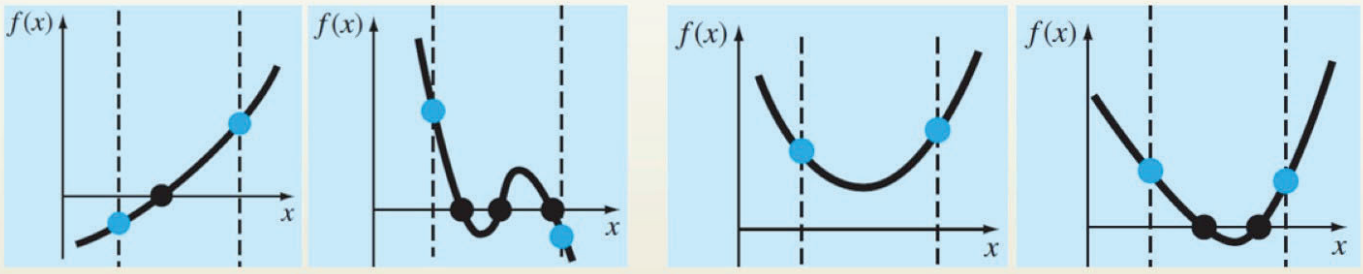
The incremental search method

- The incremental search method tests the value of the function at evenly spaced intervals and finds brackets by identifying function sign changes between neighboring points.



When does in interval contain a root?

- If $f(x_l) \times f(x_u) < 0$, that is, they have different signs (+ vs -), then the root x_r does exist inside the interval $[x_l, x_u]$



Different signs
One root

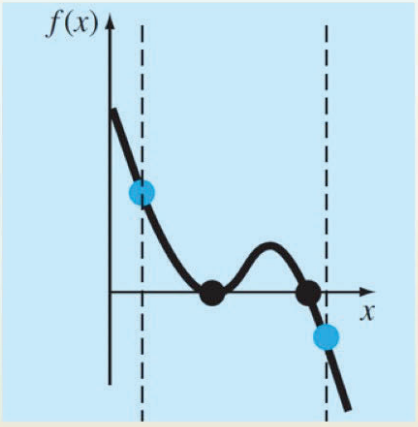
Different signs
Three roots
(odd # roots)

Same sign
No root

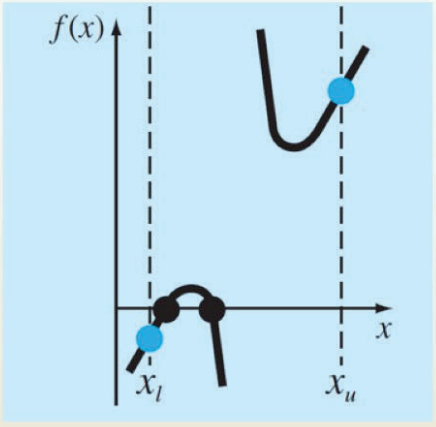
Same sign
Two roots
(even # roots)

When does in interval contain a root (special cases)

- If $f(x_l) \times f(x_u) < 0$, that is, they have different signs (+ vs -), then the root x_r does exist inside the interval $[x_l, x_u]$



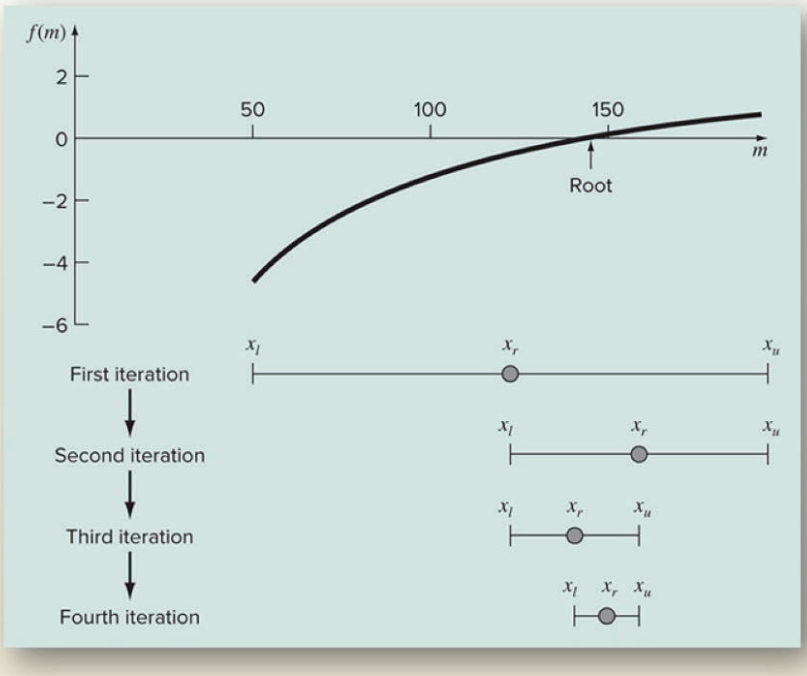
Multiple roots
e.g. $f(x) = (x-2)^2(x-4)$



Discontinuous functions

The bisection method

- After locating the interval in which the root lies
- Repeatedly divide it in half, making the interval smaller and smaller until it is small enough to meet the desire error bound

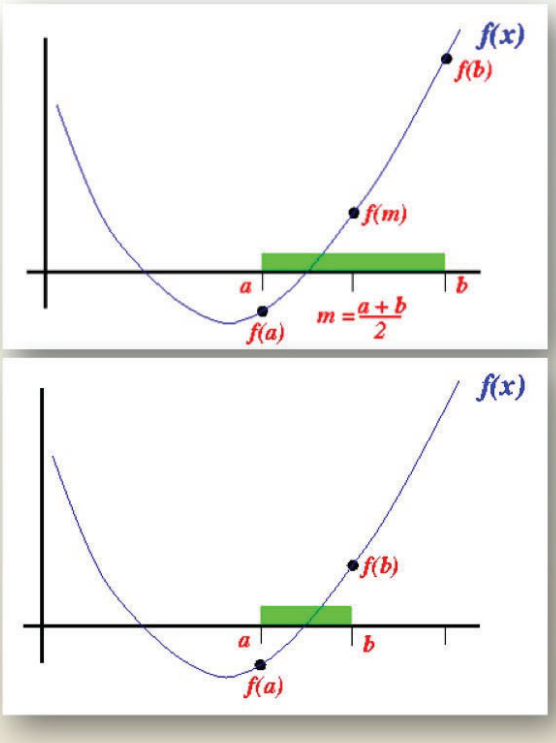


The bisection method

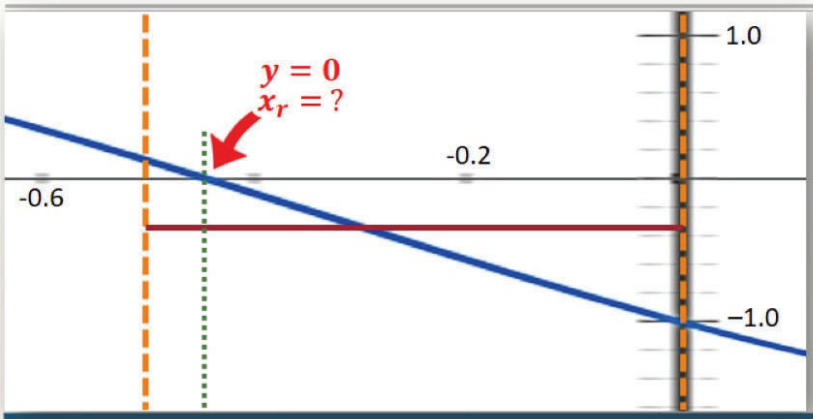
GOAL: for an eqn of 1 variable, find x such that $f(x) = 0$

- Pick lower x_l and upper x_u such that $f(x_l) \times f(x_u) < 0$
  - Use the incremental search method to find x_l and x_u
- Estimate the root = midpoint $x_r = (x_l + x_u)/2$
- Determine in which interval the root lies:
  - $f(x_l)f(x_r) < 0$ → root in the lower interval → $x_u^{new} = x_r$
  - $f(x_r)f(x_u) < 0$ → root in the upper interval → $x_l^{new} = x_r$
  - $f(x_l)f(x_r) = 0$ → the root is x_r, stop here
- Compute errors: $\varepsilon_a = \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \times 100\%$

  if $|\varepsilon_a| < \varepsilon_s$, stop, otherwise repeat the process.



https://www.cs.emory.edu/~cheung/Courses/170/Syllabus/07/bisection.html



- Use bisection method to find the root of $y = x^3 + x^2 - 2x - 1$
- Assume that we have already located the root inside the interval $[-0.5, 0]$
- Errors 3 sig figs so $\varepsilon_s = 0.05\%$

iter	x_l	x_u	x_r	$f(x_l)$	$f(x_u)$	$f(x_r)$	ε_a
1	−0.5	0					
2							
3							
4							
5							

WORKED EXAMPLES

- The third approximation to a root of the equation $f(x) = x^3 - x - 1$ in the interval [1, 2] by bisection method is ____

PRACTICE PROBLEMS

- Compute the root of the function $f(x) = xe^{-x} - 0.3$ after two iterations using bisection method in the interval [1,5]

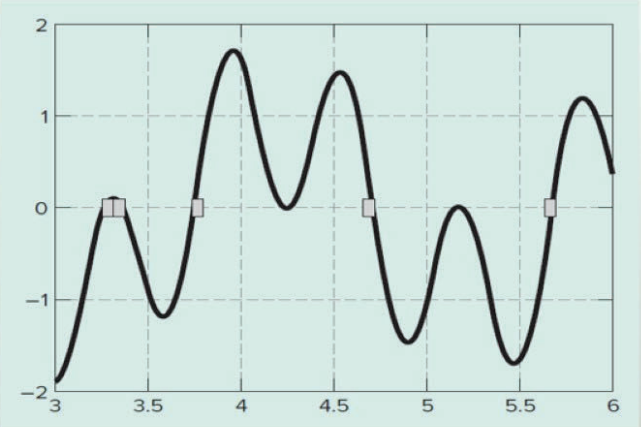
PRACTICE PROBLEMS

- Textbook Ex 5.3: the function $f(x) = \sin 10x + \cos 3x$

Use incremental search to identify brackets within the interval $[3, 6]$

Then, for each interval identified, use the bisection method to find the root

(detail solutions can be found in the textbook)



PRACTICE PROBLEMS

- Textbook Ex 5.4 – 5.5: use bisection to solve the bungee jumper problem

Find m, where $g = 9.81,\ c_d = 0.25,\ v(t=4) = 36$

Find the root in the interval $[50,200]$ and $e_s = 0.5\%$

$$v(t) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right)$$

Fill in the table.

(detail solutions can be found in the textbook)

iter	x_l	x_u	x_r	$\varepsilon_a\%$	$\varepsilon_t\%$
1					
2					
3					
4					
5					
6					
7					
8					

Incremental Search

- Find intervals that contain the roots
- *The source code is from the textbook, figure5.4*

```
import numpy as np
def incsearch(func,xmin,xmax,ns=50):
""" incsearch: incremental search locator
    incsearch(func,xmin,xmax,ns) find brackets of x that contain sign
changes in a function of x on an interval
  input:
    fun = name of the function
    xmin, xmax = endpoints of the interval
    ns = number of subintervals, default value = 50
  output:
    nb = number of bracket pairs found
    xb = list of bracket pair values
    or returns "no brackets found" """

  x = np.linspace(xmin,xmax,ns) # create array of x values
  f = [] # build array of corresponding function values
  for k in range(ns-1):
    f.append(func(x[k]))

  nb = 0
  xb = []
  for k in range(ns-2): # check adjacent pairs of function values
      if (func(x[k]) * func(x[k+1]) < 0): # for sign change
        nb = nb + 1 # increment the bracket counter
        xb.append((x[k],x[k+1])) # save the bracketing pair
  if nb==0:
    return 'no brackets found'
  else:
    return nb, xb
```

```
incsearch(lambda x: np.sin(10*x) + np.cos(3*x),3,6)
```


Bisection

- The bisection method successively narrows down search intervals that contain the root of the function $f(x)$
- *The source code is from the textbook, figure5.8*

```
def bisect(func,xl,xu,es=1.e-7,maxit=30):
""" Uses the bisection method to estimate a root of func(x). The method is iterated
until the relative error from one iteration to the next falls below the specified
value or until the maximum number of iterations is reached first.
    Input:
        func = name of the function
        xl = lower guess
        xu = upper guess
        es = relative error specification (default 1.e-7)
        maxit = maximum number of iterations allowed (default 30)
    Output:
        xm = root estimate
        fm = function value at the root estimate
        ea = actual relative error achieved
        i+1 = number of iterations required or error message if initial guesses do
not bracket solution """
    if func(xl)*func(xu)>0:
        return 'initial estimates do not bracket solution'
    xmold = xl
    for i in range(maxit):
        xm = (xl+xu)/2
        ea = abs((xm-xmold)/xm)
        if ea < es: break
        if func(xm)*func(xl)>0:
            xl = xm
        else:
            xu = xm
        xmold = xm
    return xm,func(xm),ea,i+1
```

```
bisect(lambda x: np.sin(10*x) + np.cos(3*x),xl,xu)
```